EAST 7/24/04

L Number	Hits	Search Text	DB	Time stamp
4	1	"6041406".PN.	USPAT	2004/07/24 13:10
-	19	(variable adj view) and debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 07:31
-	518	(updat$4 near3 variable) and debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/12 13:09
-	61	(updat$4 near3 variable) same debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/12 13:09
-	20	(affect$4 near3 variable) same debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/12 15:23
-	78	(watch near3 variable) same debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/12 15:27
-	74	(step$4 near3 variable) same debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/12 15:34
-	12	(choos$4 near3 variable) same debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/12 16:18
-	23	(necessar$4 near3 variable) same debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/12 17:07
-	7	("5218525" | "5230050" | "5335344" | "5410685" | "5539907" | "5732273" | "5787285").PN.	USPAT	2004/07/12 16:50
-	8	5870607.URPN.	USPAT	2004/07/12 16:52
-	145	((select$4 or determin$4) near3 variable) same debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/12 17:08
-	13	(((select$4 or determin$4) near3 variable) same debug$5) and (watch$4 near3 variable)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/12 17:09
-	9	("5119377" | "5319645" | "5526485" | "5590329" | "5608644" | "5608867" | "5784552" | "6028999" | "6094729").PN.	USPAT	2004/07/12 17:28
-	2	6295613.URPN.	USPAT	2004/07/12 17:29
-	9	("5119377" | "5319645" | "5526485" | "5590329" | "5608644" | "5608867" | "5784552" | "6028999" | "6094729").PN.	USPAT	2004/07/12 17:32

-	98	((("6412106") or ("5325530") or ("5522036") or ("5754759") or ("5815714") or ("6091896") or ("6178547") or ("6161216") or ("6016474") or ("6026362") or ("6240545") or ("6634020") or ("5778230") or ("5687375") or ("5513317") or ("5533192") or ("5784552") or ("6256777") or ("5794047") or ("5802371") or ("5970248") or ("6119206") or ("5446900") or ("5544311") or ("5870607") or ("6106572") or ("5835699") or ("5257358") or ("5956512") or ("5956479") or ("6067641") or ("6378125") or ("6587967") or ("6658649") or ("6219828") or ("6219828") or ("5619409") or ("5740440") or ("5987249") or ("6186677") or ("6233728") or ("6272672") or ("5355469") or ("5896536") or ("6263489") or ("6434741") or ("6550056") or ("5732210") or ("5784553") or ("5761408")).PN.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 14:22
-	980	debug$5 same (node)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 17:32
-	17	debug$5 same (CFG or (control adj flow adj graph))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 14:23
-	7	("5218525" | "5230050" | "5335344" | "5410685" | "5539907" | "5732273" | "5787285").PN.	USPAT	2004/07/13 14:25
-	8	5870607.URPN.	USPAT	2004/07/13 14:35
-	19	((("6412106") or ("5325530") or ("5522036") or ("5754759") or ("5815714") or ("6091896") or ("6178547") or ("6161216") or ("6016474") or ("6026362") or ("6240545") or ("6634020") or ("5778230") or ("5687375") or ("5513317") or ("5533192") or ("5784552") or ("6256777") or ("5794047") or ("5802371") or ("5970248") or ("6119206") or ("5446900") or ("5544311") or ("5870607") or ("6106572") or ("5835699") or ("5257358") or ("5956512") or ("5956479") or ("6067641") or ("6378125") or ("6587967") or ("6658649") or ("6219828") or ("6219828") or ("5619409") or ("5740440") or ("5987249") or ("6186677") or ("6233728") or ("6272672") or ("5355469") or ("5896536") or ("6263489") or ("6434741") or ("6550056") or ("5732210") or ("5784553") or ("5761408")).PN.) and (variable near3 set)	USPAT	2004/07/13 14:36
-	3507	debug$5 same (trac$4)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 17:32
-	97	debug$5 same (trac$4 near3 variable)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 17:32
-	37	debug$5 same (trac$4 near3 variable) same (breakpoint or watch$5)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 17:35

-	5	debug$5 same ((trac$4 near3 variable) near3 (select$4 or determin$4 or choosen))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 17:49
-	1	debug$5 same (trac$4 near3 (variable adj set))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 17:50
-	46	(trac$4 near3 (variable adj set))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 17:50
-	3	(trac$4 near3 (variable adj set)) and debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/13 17:50
-	18	survey near3 debug$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 07:35
-	112	(display$4 near3 variable) and (debug$4 near3 variable)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 07:36
-	48	(display$4 near3 variable) and (debug$4 near3 variable) and (determin$4 near3 variable)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 07:36
-	58	(display$4 near3 variable) and (debug$4 near3 variable) and ((determin$4 or updat$4) near3 variable)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 08:41
-	11	("4241418" | "5659484" | "5778237" | "5802290" | "5996083" | "6308311" | "6345384" | "6412061" | "6560665" | "6567837" | "6587385" | "2001/0003844" | "2003/0028690").PN.	USPAT	2004/07/14 07:38
-	4	("5652883" | "6047125" | "6260190" | "6327701").PN.	USPAT	2004/07/14 07:43
-	2	6442751.URPN.	USPAT	2004/07/14 07:44
-	0	borland near3 watch$4	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 08:41
-	1	borland same watch$4	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 08:42
-	361	watch$4 near3 updat$4	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 08:43
-	28	watch$4 near3 (updat$4 and stop$4)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 09:13

-	135	breakpoint near3 variable	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/07/14 09:13
-	1	"6041406".PN.	USPAT	2004/07/15 08:17
-	0	6678884.URPN.	USPAT	2004/07/15 08:18

CiteSeer Find: [ ] Documents Citations

Searching for **execution backtracking approach w/2 debugging**.
Restrict to: Header Title Order by: Expected citations Hubs Usage Date Try: Google (CiteSeer) Google (Web) CSB DBLP
No documents match Boolean query. Trying non-Boolean relevance query.
500 documents found. **Order: relevance to query.**

An Execution Backtracking Approach to Program Debugging - Agrawal, DeMillo, Spafford (1991) (Correct) (10 citations)
An **Execution Backtracking Approach** to Program **Debugging**
An **Execution Backtracking Approach** to Program **Debugging** Technical
hesperus.oboe.com/serc/TechReports/abstracts/catagory/../../files/TR22P.PS

Efficient Debugging with Slicing and Backtracking - Agrawal, DeMillo, Spafford (1990) (Correct) (2 citations)
It combines dynamic program slicing and **execution backtracking** techniques in a novel way. With
Efficient **Debugging** with Slicing and **Backtracking** SERC-TR-80-P Hiralal Agrawal Richard A.
of our preliminary experiment in integrating new **approach**es to software fault localization with
hesperus.oboe.com/serc/TechReports/abstracts/catagory/../../files/TR80P.PS

A short proof of Dirac's theorem on the number of edges.. - Deuber, Kostochka, Sachs (1996) (Correct)
www.mathematik.uni-bielefeld.de/sfb343/preprints/pr96067.ps.gz

A generalized collision mechanism for stochastic particle.. - Rjasanow, Wagner (Correct)
(1.12) given by Theorem 2.1. However, the **approach** to this limit depends on the choice of fl :If
www.wias-berlin.de/WIAS_publ_preprints_nr157.PS

Type Analysis for CHIP - Drabent, Pietrzak (1998) (Correct) (1 citation)
of the form of calls and successes in any **execution** of the program starting from a given class of
types of constrained atoms. Our type inference **approach** is based on bottom-up abstract interpretation,
The main intended application is program **debugging**. We consider a restricted class of
www.ipipan.waw.pl/~drabent/amast.ps.gz

Practical Estimates of the Errors Associated with the.. - Fulton, Namkung, Melvin (1992) (Correct)
repeat part of the derivation of (1) using an **approach** slightly Df 2p I A u x B v x C w x
techreports.larc.nasa.gov/pub/techreports/larc/92/conf-rpqnde-92-fulton.ps.Z

Debugging Distributed Ada Programs - Briggs, Jamieson, Randall, Wand (1994) (Correct) (1 citation)
4.2.2 Organising **execution**
"dice" and thereby further localise the bug. **Backtracking**. 1, 2, 22 This technique produces what is
and systems in general. The "traditional" **approach** to **debugging** usually involves executing the
www.cs.york.ac.uk/ftpdir/reports/YCS-94-233.ps.Z

Towards Automatic Debugging of Computer Programs - Agrawal (1991) (Correct) (15 citations)
: 18 2.9 **Execution Backtracking** :
ftp.cs.purdue.edu/pub/serc/tech-reports/By-School/Purdue/TR103P.PS.Z

The System Of Two Spinning Disks In The Torus. - Wojtkowski (1993) (Correct)
mpej.unige.ch/mp_arc/c/94/94-88.ps.gz

Minimizing Conflicts: A Heuristic Repair Method for.. - Minton, Johnston.. (1992) (Correct) (185 citations)
orders of magnitude better than traditional **backtracking** techniques. We also describe a scheduling
Abstract This paper describes a simple heuristic **approach** to solving large-scale constraint satisfaction
www.isi.edu/sims/minton/papers/aij-mc.ps

Identification Of Unknown Parameters For Heat Conductivity.. - Botkin (1995) (Correct)
heat conductivity coefficient was proposed. An **approach** related to direct methods and based on
www.appl-math.tu-muenchen.de/~botkin/hof444.ps

A Partial Approach to the Problem of Deadlocks in.. - Tricas.. (1998) (Correct)

usual in concurrent system where, even if the **execution** of each process is correct, the competition of
GISI-RR-97-05 A partial **approach** to the problem of deadlocks in processes with
www.cps.unizar.es/~ftricas/GISIRR9705.ps.gz

Uniform Reconstruction of Gaussian Processes - Müller-Gronbach, Ritter (1995) (Correct) (1 citation)
properties of the process X. In a more general **approach** we consider processes X(t) m(t) g(t)
ftp.math.fu-berlin.de/pub/math/publ/pre/1995/pr-a-95-26.ps.Z

A Practical Development Process for Parallel Large-Scale.. - Geschiere, Körver (1995) (Correct)
application in terms of subsystems and a sound **execution** order (sequential or parallel) is assumed
a practical software-engineering development **approach** build on this framework, and discusses and
ftp.ee.surrey.ac.uk/pub/research/CSRG/tech-reports/CSRG95-05.ps.Z

Effective Compiler Support for Predicated Execution .. - Mahlke, Lin, Chen, .. (1992) (Correct) (114 citations)
Effective Compiler Support for Predicated **Execution** Using the Hyperblock Scott A. Mahlke David C.
cardit.et.tudelft.nl/~steven/ilp/mahlke92.ps.gz

A Whole Sentence Maximum Entropy Language Model - Rosenfeld (1997) (Correct) (4 citations)
example of the limitations of the chain rule **approach**, consider one aspect of an sentence: its length.
www.cs.cmu.edu/afs/cs.cmu.edu/user/roni/WWW/rdi-IEEE-ASR97.ps

Articulation: An Integrated Approach to the Diagnosis.. - Peiwei Mi, Walt Scacchi (1993) (Correct) (8 citations)
software development. Use of articulation in plan **execution** supports recovery and repair of unanticipated
in order to avoid unexpected events and **backtracking**. However, they are unable to identify
Articulation: An Integrated **Approach** to the Diagnosis, Replanning, and Rescheduling
www.usc.edu/dept/ATRIUM/Papers/Articulation.ps

Running Programs Backwards: The Logical Inversion of Imperative.. - Ross (1998) (Correct) (9 citations)
Depending on the inference strategy used, **execution** of this relational program can compute the
logic programming languages incorporate **backtracking** to search for multiple solutions in its
logic programming tools. The advantage of this **approach** is that nondeterministic inversions are
www.cosc.brocku.ca/Research/TechRep/cs9403.ps

Weakly Gibbsian measures for lattice spin systems - Lörinczi, Maes (1997) (Correct)
be defined. In this paper we are taking up this **approach** and we are asking in what generality we can give
mpej.unige.ch/mp_arc/html/mp_arc/e/97-122.ps

First 20 documents Next 20

Try your query at: Google (CiteSeer) Google (Web) CSB DBLP

NPL 7/2/04

PORTAL
US Patent & Trademark Office

Subscribe (Full Service) Register (Limited Service, Free) Login

Search: The ACM Digital Library The Guide

"The program dependence graph and its use in optimization" Search

THE ACM DIGITAL LIBRARY

Feedback Report a problem Satisfaction survey

Terms used and in The program dependence graph its use optimization

Found 211 of 139,567

Sort results by relevance Save results to a Binder Try an Advanced Search

Display results expanded form Search Tips Try this search in The ACM Guide

Open results in a new window

Results 1 - 20 of 200 Result page: 1 2 3 4 5 6 7 8 9 10 next

Best 200 shown Relevance scale

1 The program dependence graph and its use in optimization
Jeanne Ferrante, Karl J. Ottenstein, Joe D. Warren
July 1987 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 9 Issue 3
Full text available: pdf(2.51 MB) Additional Information: full citation, abstract, references, citings, index terms, review

In this paper we present an intermediate program representation, called the program dependence graph (PDG), that makes explicit both the data and control dependences for each operation in a program. Data dependences have been used to represent only the relevant data flow relationships of a program. Control dependences are introduced to analogously represent only the essential control flow relationships of a program. Control dependences are derived from the ...

2 1988: Interprocedural slicing using dependence graphs
Susan Horwitz, Thomas Reps, David Binkley
April 2004 **ACM SIGPLAN Notices**, Volume 39 Issue 4
Full text available: pdf(1.95 MB) Additional Information: full citation, abstract, references

A slice of a program with respect to a program point p and variable x consists of all statements of the program that might affect the value of x at point p. This paper concerns the problem of interprocedural slicing -- generating a slice of an entire program, where the slice crosses the boundaries of procedure calls. To solve this problem, we introduce a new kind of graph to represent programs, called a *system dependence graph*, which extends previous dependence ...

3 The program dependence graph and vectorization
W. Baxter, H. R. Bauer
January 1989 **Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**
Full text available: pdf(1.04 MB) Additional Information: full citation, abstract, references, citings, index terms

Previous attempts at vectorizing programs written in a sequential high level language focused on converting control dependences to data dependences using a mechanism known as IF-conversion. After IF-conversion vector optimizations are performed on a data dependence graph. However, IF-conversion is an irrevocable process which can introduce high run-time overhead if the input program is not amenable to vectorization. This paper

uses a program dependence graph as the intermediate r ...

4 Program optimization and parallelization using idioms
Shlomit S. Pinter, Ron Y. Pinter
January 1991 **Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**
Full text available: pdf(1.23 MB) Additional Information: full citation, references, citings, index terms

5 Automatic construction of sparse data flow evaluation graphs
Jong-Deok Choi, Ron Cytron, Jeanne Ferrante
January 1991 **Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**
Full text available: pdf(1.00 MB) Additional Information: full citation, references, citings, index terms

6 A program integration algorithm that accommodates semantics-preserving transformations
Wuu Yang, Susan Horwitz, Thomas Reps
October 1990 **ACM SIGSOFT Software Engineering Notes , Proceedings of the fourth ACM SIGSOFT symposium on Software development environments**, Volume 15 Issue 6
Full text available: pdf(1.31 MB) Additional Information: full citation, abstract, references, citings, index terms

Given a program Base and two variants, A and B, each created by modifying separate copies of Base, the goal of program integration is to determine whether the modifications interfere, and if they do not, to create an integrated program that includes both sets of changes as well as the portions of Base preserved in both variants. Text-based integration techniques, such as the one used by the UNIX dif ...

7 Generating fast code from concurrent program dependence graphs
Jia Zeng, Cristian Soviani, Stephen A. Edwards
June 2004 **ACM SIGPLAN Notices , Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools**, Volume 39 Issue 7
Full text available: pdf(90.25 KB) Additional Information: full citation, abstract, references, index terms

While concurrency in embedded systems is most often supplied by real-time operating systems, this approach can be unpredictable and difficult to debug. Synchronous concurrency, in which a system marches in lockstep to a global clock, is conceptually easier and potentially more efficient because it can be statically scheduled beforehand.We present an algorithm for generating efficient sequential code from such synchronous concurrent specifications. Starting from a concurrent program dependence gr ...

Keywords: concurrent, esterel, program dependence graph, sequencial

8 Balancing register allocation across threads for a multithreaded network processor
Xiaotong Zhuang, Santosh Pande
June 2004 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**, Volume 39 Issue 6
Full text available: pdf(429.85 KB) Additional Information: full citation, abstract, references, index terms

Modern network processors employ multi-threading to allow concurrency amongst multiple packet processing tasks. We studied the properties of applications running on the network

processors and observed that their imbalanced register requirements across different threads at different program points could lead to poor performance. Many times application needs demand some threads to be more performance critical than others and thus by controlling the register allocation across threads one could impa ...

Keywords: multithreaded processor, network processor, register allocation

9 Context-sensitive slicing of concurrent programs
Jens Krinke
September 2003 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering**, Volume 28 Issue 5
Full text available: pdf(514.28 KB) Additional Information: full citation, abstract, references, index terms

Program slicing is a technique to identify statements that may influence the computations at other statements. Precise slicing has been shown to be undecidable for concurrent programs. This work presents the first context-sensitive approach to slice concurrent programs accurately. It extends the well known structures of the control flow graph and the (interprocedural) program dependence graph for concurrent programs with interference. This new technique does not require serialization or inlining ...

Keywords: concurrency, context-sensitive, parallelism, program analysis, program slicing

10 Instruction reordering for fork-join parallelism
V. Sarkar
June 1990 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1990 conference on Programming language design and implementation**, Volume 25 Issue 6
Full text available: pdf(1.02 MB) Additional Information: full citation, references, citings, index terms

11 The program dependence web: a representation supporting control-, data-, and demand-driven interpretation of imperative languages
Karl J. Ottenstein, Robert A. Ballance, Arthur B. MacCabe
June 1990 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1990 conference on Programming language design and implementation**, Volume 25 Issue 6
Full text available: pdf(1.59 MB) Additional Information: full citation, abstract, references, citings, index terms

The Program Dependence Web (PDW) is a program representation that can be directly interpreted using control-, data-, or demand-driven models of execution. A PDW combines a single-assignment version of the program with explicit operators that manage the flow of data values. The PDW can be viewed as an augmented Program Dependence Graph. Translation to the PDW representation provides the basis for projects to compile Fortran onto dynamic dataflow architectures and simulators. ...

12 Technical contributions: Intermediate program representations in compiler construction: a supplemental bibliography
Karl J. Ottenstein
July 1984 **ACM SIGPLAN Notices**, Volume 19 Issue 7
Full text available: pdf(197.96 KB) Additional Information: full citation, references

13 Tutorial: Compiling concurrent languages for sequential processors
Stephen A. Edwards
April 2003 **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, Volume 8 Issue 2
Full text available: pdf(771.65 KB) Additional Information: full citation, abstract, references, index terms, review

Embedded systems often include a traditional processor capable of executing sequential code, but both control and data-dominated tasks are often more naturally expressed using one of the many domain-specific concurrent specification languages. This article surveys a variety of techniques for translating these concurrent specifications into sequential code. The techniques address compiling a wide variety of languages, ranging from dataflow to Petri nets. Each uses a different method, to some degr ...

Keywords: Compilation, Esterel, Lustre, Petri nets, Verilog, code generation, communication, concurrency, dataflow, discrete-event, partial evaluation, sequential

14 An efficient method of computing static single assignment form
R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, F. K. Zadeck
January 1989 **Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**
Full text available: pdf(1.24 MB) Additional Information: full citation, references, citings, index terms

15 Automatic generation of DAG parallelism
R. Cytron, M. Hind, W. Hsieh
June 1989 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation**, Volume 24 Issue 7
Full text available: pdf(1.58 MB) Additional Information: full citation, references, citings, index terms

16 The semantics of program dependence
E. Cartwright, M. Felleisen
June 1989 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation**, Volume 24 Issue 7
Full text available: pdf(1.31 MB) Additional Information: full citation, abstract, references, citings, index terms

Optimizing and parallelizing compilers for procedural languages rely on various forms of program dependence graphs (pdgs) to express the essential control and data dependencies among atomic program operations. In this paper, we provide a semantic justification for this practice by deriving two different forms of program dependence graph — the output pdg and the def-order pdg—and their semantic definitions from non-strict ...

17 On the adequacy of program dependence graphs for representing programs
S. Horwitz, J. Prins, T. Reps
January 1988 **Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**
Full text available: pdf(1.56 MB) Additional Information: full citation, abstract, references, citings, index terms

Program dependence graphs were introduced by Kuck as an intermediate program representation well suited for performing optimizations, vectorization, and parallelization. There are also additional applications for them as an internal program representation in program development environments. In this paper we examine the issue of whether a

program dependence graph is an adequate structure for representing a program's execution behavior. (This question has apparently never been add ...

18 Static analysis of low-level synchronization
David Callahan, Jaspal Sublok
November 1988 **ACM SIGPLAN Notices , Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on Parallel and distributed debugging**, Volume 24 Issue 1
Full text available: pdf(990.97 KB) Additional Information: full citation, references, citings, index terms

19 Automated method-extraction refactoring by using block-based slicing
Katsuhisa Maruyama
May 2001 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 2001 symposium on Software reusability: putting software reuse in context**, Volume 26 Issue 3
Full text available: pdf(174.09 KB) Additional Information: full citation, abstract, references, index terms

Refactoring improves the design of existing code but is not complex to do by hand. This paper proposes a mechanism that automatically refactors methods of object-oriented programs by using program slicing. To restructure a method without changing its observable behavior, the mechanism uses block-based slicing that does not extract the fragments of code from the whole program but from the region consisting of some consecutive basic-blocks of the program. A refactoring tool implementing the m ...

20 Interprocedural control dependence
Saurabh Sinha, Mary Jean Harrold, Gregg Rothermel
April 2001 **ACM Transactions on Software Engineering and Methodology (TOSEM)**, Volume 10 Issue 2
Full text available: pdf(506.53 KB) Additional Information: full citation, abstract, references, citings, index terms

Program-dependence information is useful for a variety of applications, such as software testing and maintenance tasks, and code optimization. Properly defined, control and data dependences can be used to identify semantic dependences. To function effectively on whole programs, tools that utilize dependence information require information about interprocedural dependences: dependences that are identified by analyzing the interactions among procedures. Many techniques for computing interproc ...

Keywords: interprocedural analysis, interprocedural control dependence, program slicing, semantic dependence, software maintenance

CiteSeer Find: debugging Documents Citations

Searching for **PHRASE debugging variable set.**
Restrict to: Header Title Order by: Expected citations Hubs Usage Date Try: Google (CiteSeer) Google (Web) CSB DBLP
No documents match Boolean query. Trying non-Boolean relevance query.
500 documents found. **Order: relevance to query.**

Measuring Design-level Cohesion - Bieman (1998) (Correct) (3 citations)
properties of implementations such as "ease of **debugging**, ease of maintenance, and ease of modification" methods through common references to instance **variables**[1, 10, 11]Method bodies are needed to apply by skilled engineers. These engineers would apply a **set** of subjective criteria to analyze associations
ftp.cs.colostate.edu/pub/bieman/tse98.ps.Z

Meaning Extraction of Single Facts: A Quantitative Analysis - Bagga (1996) (Correct)
the examples were made for the purpose of **debugging**. In other cases, the examples came from pieces 50 List of Figures 1.1 A Semantic Network **Variable** :6 1.2 The
is a 4-tuple (N T LN L T)where N is the **set** of nodes, T is the **set** of transitions (relations)
www.cs.duke.edu/~amit/prelims.ps.gz

Design-For-Debugging of Application Specific Designs - Miodrag Potkonjak Sujit (1995) (Correct) (4 citations)
Design-For-**Debugging** of Application Specific Designs Miodrag
and observability of the user specified **variables**. Two key conceptually new design ideas that
www.cs.ucla.edu/~miodrag/papers/Potkonjak_ICCAD_95.pdf

Closed-Form Mapping Conditions for the Synthesis of Linear.. - Xue (1995) (Correct)
every time when it is applied to individual **variables**. The closed-form condition was given in the were made in [29]Lee and Kedem [29] gave a **set** of necessary and sufficient conditions on mappings feasible. Shang and Fortes [24] presented a **set** of closed-form conditions on computational
cs.une.edu.au/~xue/paper/jvsp95.ps.Z

Heuristics for Automatic Localization of Software Faults - Pan, Spafford (1992) (Correct)
July 29, 1992 Keywords: software testing, **debugging**, dynamic program slicing, fault localization, (3) making hypotheses about suspicious faults (**variables** and locations)and (4) restoring program state
ftp.cs.purdue.edu/pub/serc/tech-reports/By-School/Purdue/TR116P.PS.Z

A New Characterization of Lambda Definability - Jung, Tiuryn (1993) (Correct) (15 citations)
Henkin models remains an open problem. But if the **variable set** allowed in terms is also restricted to be models using logical relations defined over ordered **sets** with varying arity. The advantage of this over models remains an open problem. But if the **variable set** allowed in terms is also restricted to be finite
www.cs.bham.ac.uk/~axj/pub/papers/log-rel.ps.gz

The Mathematics of Set Predicates in Prolog - Börger, Rosenzweig (1993) (Correct) (1 citation)
(abstraction,collection) operator to **variable** x and expression P (x)which binds all
The Mathematics of Set Predicates in Prolog Egon B orger 1 Doan
Abstract. We provide a logical specification of **set** predicates findall and bagof of Prolog. The
linux.eecs.umich.edu/.5/groups/Ealgebras/bagof.ps.gz

Debugging Distributed Ada Programs - Briggs, Jamieson, Randall, Wand (1994) (Correct) (1 citation)
Debugging Distributed Ada Programs J S Briggs S D
www.cs.york.ac.uk/ftpdir/reports/YCS-94-233.ps.Z

Leader of Group : - Contributors Unl (Correct)
the design and implementation of a monitoring and **debugging** architecture supporting the functionalities allowing the interactive display of process **variables**. The prototype runs on the PVM environment, and **debugger** instance. The master daemon and the **set** of local daemons form a distributed architecture
ftp.cpc.wmin.ac.uk/pub/sepp/P6/task5.ps.gz

A Guide To The NU-Prolog Debugging Environment - Bert Thompson (Correct)
A Guide To The NU-Prolog **Debugging** Environment Bert Thompson and Lee Naish
www.cs.mu.oz.au/publications/tr_db/./mu_96_38.ps.gz

Constraint-Directed Backtracking Algorithm for.. - Pang, Goodwin (1998) (Correct)
(BT) share a similar style of instantiating **variables** (forward) and re-instantiating **variables** searches instantiations of **variables** in a **variable set** from a given constraint posed on that **variable set** set from a given constraint posed on that **variable set** and appends it to a partial solution, whereas BT
www.cs.uregina.ca/~pang/rp96.ps.Z

Making Real-Time Reactive Systems Reliable - Marzullo, Wood (1991) (Correct) (12 citations)
reactive systems, but so are monitoring and **debugging** systems and distributed application management machine load, allocated resources, and the global **variables** of running programs. Second, the programmer system specification in terms of time that is, to **set** hard real-time constraints on the execution of the
ftp.cs.ucsd.edu/pub/faculty/marzullo/TR90-1155.ps.Z

Fault Investigation and Trial - Viravan (1991) (Correct)
bugs is one of the most time-consuming tasks in **debugging**. Though external resources, such as explicit
ftp.cs.purdue.edu/pub/serc/tech-reports/By-Project/debugger/TR104P.PS.Z

Strategic Oscillation in Heuristic Local Search - Laurent Mynard (Correct)
its complementary. Theorem 1. The lower bound of **variable** x i is 0. The upper bound of **variable** x i is x i solutions should be instead of exploring the whole **set** almost randomly as other local search algorithms of this approach. 2 Definitions Let S be a discrete **set**, called exploration **set**. It is partitionned into
www-poleia.lip6.fr/~mynard/frames/../ps/kbcs96.ps.gz

Cross-Validated C4.5: Using Error Estimation for Automatic.. - John (1994) (Correct)
by a vector of attribute (feature, measurement **variable**) values ¬x and a categorical class label y issue in the use of these algorithms is how to **set** the parameters of the algorithm. While the default accuracy of the induced trees on independent test **sets** is generally higher than the accuracy when using
elib.stanford.edu/pub/reports/cs/tn/94/12/CS-TN-94-12.ps

Relative Debugging Using Multiple Program Versions - Abramson, Sosic (Correct)
Relative **Debugging** Using Multiple Program Versions David
www.rdt.monash.edu.au/~davida/papers/islip.ps.Z

Regular Expressions with Nested Levels of Back Referencing Form a .. - Larsen (1997) (Correct) (1 citation)
construct. If a subexpression is named by some **variable**, whenever this subexpression matches some [1]Let \Sigma, the alphabet, be an infinite 1 **set** of symbols, and let \Psi be an infinite **set** of 1 **set** of symbols, and let \Psi be an infinite **set** of names. The following grammar defines the syntax
ftp.imada.ou.dk/pub/papers/pp-1997/13.ps.gz

A Comparison of Constant and Variable Amplitude Command Shaping.. - Lucy Pao (1995) (Correct)
September 1995 A Comparison of Constant and **Variable** Amplitude Command Shaping Techniques for schemes. The input shaper is determined from a **set** of equations that constrain the dynamic response on the impulse amplitudes [1, 6, 7, 8, 12, 14]The **set** of constraint equations governing the dynamic
schof.colorado.edu/~pao/anonftp/cca95capvap.ps

Enhancing Debugging Technology - Viravan (1994) (Correct)
Enhancing **Debugging** Technology A Thesis Submitted To The Faculty
ftp.cs.purdue.edu/pub/serc/tech-reports/By-School/Purdue/TR151P.PS.Z

Type Analysis for CHIP - Drabent, Pietrzak (1998) (Correct) (1 citation)
The main intended application is program **debugging**. We consider a restricted class of the semantics of a program. In particular, **variables** are typed and may only be bound to the values logic programs wrt to a specification, which gives a **set** of procedure calls and a **set** of procedure
www.ipipan.waw.pl/~drabent/amast.ps.gz

First 20 documents Next 20

Try your query at: Google (CiteSeer) Google (Web) CSB DBLP

Terms used **A Mechanism for Efficient Debugging of Parallel Programs**

Found **11** of **139,567**

Sort results by relevance

Display results expanded form

Save results to a Binder

Search Tips

Open results in a new window

Try an Advanced Search

Try this search in The ACM Guide

Results 1 - 11 of 11

Relevance scale

1 1989: On-the-fly detection of access anomalies

Edith Schonberg

April 2004 **ACM SIGPLAN Notices**, Volume 39 Issue 4

Full text available: pdf(1.39 MB) Additional Information: full citation, abstract, references

Access anomalies are a common class of bugs in shared-memory parallel programs. An access anomaly occurs when two concurrent execution threads both write (or one thread reads and the other writes) the same shared memory location without coordination. Approaches to the detection of access anomalies include static analysis, post-mortem trace analysis, and on-the-fly monitoring.A general on-the-fly algorithm for access anomaly detection is presented, which can be applied to programs with both neste ...

2 Optimally profiling and tracing programs

Thomas Ball, James R. Larus

February 1992 **Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

Full text available: pdf(1.27 MB) Additional Information: full citation, abstract, references, citings, index terms

This paper presents algorithms for inserting monitoring code to profile and trace programs. These algorithms greatly reduce the cost of measuring programs. Profiling counts the number of times each basic block in a program executes and has a variety of applications. Instruction traces are the basis for trace-driven simulation and analysis, and are also used in trace-driven debugging. The profiling algorithm chooses a placement of counters that is optimized—and frequently op ...

3 Demonic memory for process histories

P. R. Wilson, T. G. Moher

June 1989 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation**, Volume 24 Issue 7

Full text available: pdf(1.55 MB) Additional Information: full citation, abstract, references, citings, index terms

Demonic memory is a form of reconstructive memory for process histories. As a process executes, its states are regularly checkpointed, generating a history of the process at low time resolution. Following the initial generation, any prior state of the process can be reconstructed by starting from a checkpointed state and re-executing the process up through the desired state, thereby exploiting the redundancy between the states of a process and the description of that process (i.e., a comput ...

4 A mechanism for efficient debugging of parallel programs

Barton P. Miller, Jong-Deok Choi

November 1988 **ACM SIGPLAN Notices , Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on Parallel and distributed debugging**, Volume 24 Issue 1

Full text available: pdf(1.22 MB) Additional Information: full citation, abstract, references, citings, index terms, review

This paper addresses the design and implementation of an integrated debugging system for parallel programs running on shared memory multi-processors (SMMP). We describe the use of flowback analysis to provide information on causal relationships between events in a program's execution without re-executing the program for debugging. We introduce a mechanism called incremental tracing that, by using semantic analyses of the debugged program, makes the flowback ...

5 On-the-fly detection of access anomalies

D. Schonberg

June 1989 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation**, Volume 24 Issue 7

Full text available: pdf(1.26 MB) Additional Information: full citation, abstract, references, citings, index terms

Access anomalies are a common class of bugs in shared-memory parallel programs. An access anomaly occurs when two concurrent execution threads both write (or one thread reads and the other writes) the same shared memory location without coordination. Approaches to the detection of access anomalies include static analysis, post-mortem trace analysis, and on-the-fly monitoring. A general on-the-fly algorithm for access anomaly detection is presented, which can be applied to program ...

6 Supporting reverse execution for parallel programs

Douglas Z. Pan, Mark A. Linton

November 1988 **ACM SIGPLAN Notices , Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on Parallel and distributed debugging**, Volume 24 Issue 1

Full text available: pdf(659.72 KB) Additional Information: full citation, abstract, references, citings, index terms

Parallel programs are difficult to debug because they run for a, long time and two executions may yield different results. Reverse execution, is a simple and powerful concept that solves both these problems. We are designing a tool for debugging parallel programs, called Recap, that provides the illusion of reverse execution using checkpoints and event recording and playback. During normal execution, Recap logs the results of system calls and shared memory reads: as well as ...

7 A mechanism for efficient debugging of parallel programs

B. P. Miller, Jong-Deok Choi

June 1988 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation**, Volume 23 Issue 7

Full text available: pdf(1.29 MB) Additional Information: full citation, abstract, references, citings, index terms

This paper addresses the design and implementation of an integrated debugging system for parallel programs running on shared memory multi-processors (SMMP). We describe the use of flowback analysis to provide information on causal relationships between events in a program's execution without re-executing the program for debugging. We introduce a mechanism called incremental tracing that, by using semantic analyses of the debugged program, makes the flowback ...

8 Optimal tracing and incremental reexecution for debugging long-running programs
Robert H. B. Netzer, Mark H. Weaver
June 1994 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation**, Volume 29 Issue 6
Full text available: pdf(1.34 MB) Additional Information: full citation, references, citings, index terms

9 Loop monotonic computations: an approach for the efficient run-time detection of races
Rajiv Gupta, Madalene Spezialetti
September 1991 **Proceedings of the symposium on Testing, analysis, and verification**
Full text available: pdf(1.20 MB) Additional Information: full citation, references, citings, index terms

10 Improving the accuracy of data race detection
Robert H. B. Netzer, Barton P. Miller
April 1991 **ACM SIGPLAN Notices , Proceedings of the third ACM SIGPLAN symposium on Principles and practice of parallel programming**, Volume 26 Issue 7
Full text available: pdf(1.37 MB) Additional Information: full citation, references, citings, index terms

11 Detecting access anomalies in programs with critical sections
Anne Dinning, Edith Schonberg
December 1991 **ACM SIGPLAN Notices , Proceedings of the 1991 ACM/ONR workshop on Parallel and distributed debugging**, Volume 26 Issue 12
Full text available: pdf(931.91 KB) Additional Information: full citation, references, citings, index terms

Results 1 - 11 of 11

CiteSeer Find: Documents Citations

Searching for **a mechanism w/2 efficient debugging w/2 parallel programs**.
Restrict to: Header Title Order by: Expected citations Hubs Usage Date Try: Google (CiteSeer) Google (Web) CSB DBLP
2 documents found. **Order: number of citations.**

nCUBE 2 Processor Manual Rel. - Ncub Ncube Corporation (Correct)
Dec. 1989 [MiCh88] B.P. Miller, J.D. Choi, **A Mechanism** for **Efficient Debugging** of **Parallel Programs**"
B.P. Miller, J.D. Choi, **A Mechanism** for **Efficient Debugging** of **Parallel Programs**"Proc.
Choi, **A Mechanism** for **Efficient Debugging** of **Parallel Programs**"Proc. SIGPLAN/SIGOPS Workshop on Parallel
zeus.gup.uni-linz.ac.at/techrep/postscript/ACPC/TR_95_3.ps.gz

EMU - Event Monitoring Utility - Grabner, Kranzmüller, Volkert (Correct)
Dec. 1989 [MiCh88] B.P. Miller, J.D. Choi, **A Mechanism** for **Efficient Debugging** of **Parallel Programs**"
B.P. Miller, J.D. Choi, **A Mechanism** for **Efficient Debugging** of **Parallel Programs**"Proc.
Choi, **A Mechanism** for **Efficient Debugging** of **Parallel Programs**"Proc. SIGPLAN/SIGOPS Workshop on Parallel
ftp.risc.uni-linz.ac.at/pub/acpc/reports/acpc.95-b.ps.gz

Try your query at: Google (CiteSeer) Google (Web) CSB DBLP

Terms used **Interprocedural slicing using dependence graphs** Found **105** of **139,567**

Sort results by relevance Save results to a Binder Try an Advanced Search
Display results expanded form Search Tips Try this search in The ACM Guide
Open results in a new window

Results 1 - 20 of 105 Result page: **1** 2 3 4 5 6 next

Relevance scale

1 1988: Interprocedural slicing using dependence graphs
Susan Horwitz, Thomas Reps, David Binkley
April 2004 **ACM SIGPLAN Notices**, Volume 39 Issue 4
Full text available: pdf(1.05 MB) Additional Information: full citation, abstract, references

A slice of a program with respect to a program point *p* and variable *x* consists of all statements of the program that might affect the value of *x* at point *p*. This paper concerns the problem of interprocedural slicing -- generating a slice of an entire program, where the slice crosses the boundaries of procedure calls. To solve this problem, we introduce a new kind of graph to represent programs, called a *system dependence graph*, which extends previous dependence ...

2 Interprocedural slicing using dependence graphs
Susan Horwitz, Thomas Reps, David Binkley
January 1990 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 12 Issue 1
Full text available: pdf(2.69 MB) Additional Information: full citation, abstract, references, citings, index terms, review

The notion of a program slice, originally introduced by Mark Weiser, is useful in program debugging, automatic parallelization, and program integration. A slice of a program is taken with respect to a program point p and a variable x; the slice consists of all statements of the program that might affect the value of x at point p. This paper concerns the problem of interprocedural slicing—generating a ...

3 Interprocedural slicing using dependence graphs
S. Horwitz, T. Reps, D. Binkley
June 1988 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation**, Volume 23 Issue 7
Full text available: pdf(1.25 MB) Additional Information: full citation, abstract, references, citings, index terms

A slice of a program with respect to a program point p and variable x consists of all statements of the program that might affect the value of x at point p. This paper concerns the problem of interprocedural slicing — generating a slice of an entire program, where the slice crosses the boundaries of procedure calls. To solve this problem, we introduce a new kind of graph to represent programs, called a system ...

4

Interconvertbility of set constraints and context-free language reachability
David Melski, Thomas Reps
December 1997 **ACM SIGPLAN Notices , Proceedings of the 1997 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation**, Volume 32 Issue 12
Full text available: pdf(1.56 MB) Additional Information: full citation, abstract, references, citings, index terms

We show the interconvertibility of context-free-language reachability problems and a class of set-constraint problems: given a context-free-language reachability problem, we show how to construct a set-constraint problem whose answer gives a solution to the reachability problem; given a set-constraint problem, we show how to construct a context-free-language reachability problem whose answer gives a solution to the set-constraint problem. The interconvertibility of these two formalisms offers an ...

5 Pointer-induced aliasing: a problem taxonomy
William Landi, Barbara G. Ryder
January 1991 **Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**
Full text available: pdf(846.20 KB) Additional Information: full citation, references, citings, index terms

6 Automatic high-quality reengineering of database programs by abstraction, transformation and reimplementation
Yossi Cohen, Yishai A. Feldman
July 2003 **ACM Transactions on Software Engineering and Methodology (TOSEM)**, Volume 12 Issue 3
Full text available: pdf(245.87 KB) Additional Information: full citation, abstract, references, index terms

Old-generation database models, such as the indexed-sequential, hierarchical, or network models, provide record-level access to their data, with all application logic residing in the hosting program. In contrast, relational databases can perform complex operations, such as filter, aggregation, and join, on multiple records without an external specification of the record-access logic. Programs written for relational databases attempt to move as much of the application logic as possible into the d ...

Keywords: Database program reengineering, query graphs, temporal abstraction, the plan calculus

7 Context-sensitive slicing of concurrent programs
Jens Krinke
September 2003 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering**, Volume 28 Issue 5
Full text available: pdf(514.28 KB) Additional Information: full citation, abstract, references, index terms

Program slicing is a technique to identify statements that may influence the computations at other statements. Precise slicing has been shown to be undecidable for concurrent programs. This work presents the first context-sensitive approach to slice concurrent programs accurately. It extends the well known structures of the control flow graph and the (interprocedural) program dependence graph for concurrent programs with interference. This new technique does not require serialization or inlining ...

Keywords: concurrency, context-sensitive, parallelism, program analysis, program slicing

8 Analysis and verification: Buffer overrun detection using linear programming and static analysis
Vinod Ganapathy, Somesh Jha, David Chandler, David Melski, David Vitek
October 2003 **Proceedings of the 10th ACM conference on Computer and communication security**
Full text available: pdf(190.29 KB) Additional Information: full citation, abstract, references, index terms

This paper addresses the issue of identifying buffer overrun vulnerabilities by statically analyzing C source code. We demonstrate a light-weight analysis based on modeling C string manipulations as a linear program. We also present fast, scalable solvers based on linear programming, and demonstrate techniques to make the program analysis context sensitive. Based on these techniques, we built a prototype and used it to identify several vulnerabilities in popular security critical applications.

Keywords: buffer overruns, linear programming, static analysis

9 Dependence analysis for pointer variables
S. Horwitz, P. Pfeiffer, T. Reps
June 1989 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation**, Volume 24 Issue 7
Full text available: pdf(1.49 MB) Additional Information: full citation, abstract, references, citings, index terms

Our concern is how to determine data dependencies between program constructs in programming languages with pointer variables. We are particularly interested in computing data dependencies for languages that manipulate heap-allocated storage, such as Lisp and Pascal. We have defined a family of algorithms that compute safe approximations to the flow, output, and anti-dependencies of a program written in such a language. Our algorithms account for destructive updates to fields of a structure ...

10 Special issue on prototypes of deductive database systems: The CORAL deductive system
Raghu Ramakrishnan, Divesh Srivastava, S. Sudarshan, Praveen Seshadri
April 1994 **The VLDB Journal — The International Journal on Very Large Data Bases**, Volume 3 Issue 2
Full text available: pdf(3.03 MB) Additional Information: full citation, abstract, references, citings

CORAL is a deductive system that supports a rich declarative language, and an interface to C++, which allows for a combination of declarative and imperative programming. A CORAL declarative program can be organized as a collection of interacting modules. CORAL supports a wide range of evaluation strategies, and automatically chooses an efficient strategy for each module in the program. Users can guide query optimization by selecting from a wide range of control choices. The CORAL system provides ...

Keywords: deductive database, logic programming system, query language

11 Session 4: static program analysis: Improving program slicing with dynamic points-to data
Markus Mock, Darren C. Atkinson, Craig Chambers, Susan J. Eggers
November 2002 **ACM SIGSOFT Software Engineering Notes**, Volume 27 Issue 6
Full text available: pdf(1.05 MB) Additional Information: full citation, abstract, references, index terms

Program slicing is a potentially useful analysis for aiding program understanding. However,

slices of even small programs are often too large to be generally useful. Imprecise pointer analyses have been suggested as one cause of this problem. In this paper, we use dynamic points-to data, which represents optimal or optimistic pointer information, to obtain a bound on the best case slice size improvement that can be achieved with improved pointer precision. Our experiments show that slice size ca ...

Keywords: dynamic analysis, points-to analysis, program slicing

12 From symptom to cause: localizing errors in counterexample traces
Thomas Ball, Mayur Naik, Sriram K. Rajamani
January 2003 **ACM SIGPLAN Notices , Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**, Volume 38 Issue 1
Full text available: pdf(208.52 KB) Additional Information: full citation, abstract, references, index terms

There is significant room for improving users' experiences with model checking tools. An error trace produced by a model checker can be lengthy and is indicative of a *symptom* of an error. As a result, users can spend considerable time examining an error trace in order to understand the *cause* of the error. Moreover, even state-of-the-art model checkers provide an experience akin to that provided by parsers before syntactic error recovery was invented: they report a single error trac ...

Keywords: debugging, software model checking

13 Static program analysis: Improving program slicing with dynamic points-to data
Markus Mock, Darren C. Atkinson, Craig Chambers, Susan J. Eggers
November 2002 **Proceedings of the tenth ACM SIGSOFT symposium on Foundations of software engineering**
Full text available: pdf(109.02 KB) Additional Information: full citation, abstract, references, citings, index terms

Program slicing is a potentially useful analysis for aiding program understanding. However, slices of even small programs are often too large to be generally useful. Imprecise pointer analyses have been suggested as one cause of this problem. In this paper, we use dynamic points-to data, which represents optimal or optimistic pointer information, to obtain a bound on the best case slice size improvement that can be achieved with improved pointer precision. Our experiments show that slice size ca ...

Keywords: dynamic analysis, points-to analysis, program slicing

14 Session S6.2: compilers and program analysis: Scenario-based software characterization as a contingency to traditional program profiling
Jeffry T. Russell, Margarida F. Jacome
October 2002 **Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems**
Full text available: pdf(142.05 KB) Additional Information: full citation, abstract, references, citings, index terms

Program profiling is common way to characterize program behavior based on representative input. Some software, especially in embedded systems, cannot be profiled do to lack of tools or problems introduced by instrumentation of the code. As an alternative to traditionally profiling, a static analysis technique is proposed that allows a designer to characterize the flow of control of software.Operating on a flow graph representation of software, the proposed technique assists an expert designer in ...

Keywords: constraint, control flow, embedded system, performance, predicate, profiling, program profile, scenario, static analysis, typical behavior

15 Technical papers: software maintenance: Concern graphs: finding and describing concerns using structural program dependencies
Martin P. Robillard, Gail C. Murphy
May 2002 **Proceedings of the 24th International conference on Software engineering**

Full text available: pdf(1.35 MB) Additional Information: full citation, abstract, references, citings, index terms

Many maintenance tasks address concerns, or features, that are not well modularized in the source code comprising a system. Existing approaches available to help software developers locate and manage scattered concerns use a representation based on lines of source code, complicating the analysis of the concerns. In this paper, we introduce the Concern Graph representation that abstracts the implementation details of a concern and makes explicit the relationships between different parts of the co ...

16 Technical papers: program analysis: An empirical study of predicate dependence levels and trends
David Binkley, Mark Harman
May 2003 **Proceedings of the 25th international conference on Software engineering**

Full text available: pdf(851.32 KB) Publisher Site Additional Information: full citation, abstract, references

Many source code analyses are closely related to and strongly influenced by interdependence among program components. This paper reports results from an empirical study of the interdependences involving program predicates and the formal parameters and global variables which potentially affect them.The findings show that it is possible to eliminate from consideration approximately 30% of the formal parameters, 50% of the 'touched' global variables, and 97% of the 'visible' global variables.Anothe ...

Keywords: Dependence Analysis, Program Slicing

17 Equivalence analysis and its application in improving the efficiency of program slicing
Donglin Liang, Mary Jean Harrold
July 2002 **ACM Transactions on Software Engineering and Methodology (TOSEM)**, Volume 11 Issue 3

Full text available: pdf(457.78 KB) Additional Information: full citation, abstract, references, index terms

Existing methods for handling pointer variables during dataflow analyses can make such analyses inefficient in both time and space because the data-flow analyses must store and propagate large sets of data facts that are introduced by dereferences of pointer variables. This article presents *equivalence analysis*, a general technique to improve the efficiency of data-flow analyses in the presence of pointer variables. The technique identifies equivalence relations among the memory locations ...

Keywords: Alias analysis, data-flow analysis, program slicing

18 On optimal slicing of parallel programs
Markus Müller-Olm, Helmut Seidl
July 2001 **Proceedings of the thirty-third annual ACM symposium on Theory of computing**

Additional Information:

Full text available: pdf(225.69 KB) full citation, abstract, references, citings, index terms

Optimal program slicing determines for a statement S in a program &pgr; whether or not S affects a specified set of statements, given that all conditionals in &pgr; are interpreted as non-deterministic choices.Only recently, it has been shown that reachability of program points and hence also optimal slicing is undecidable for multi-threaded programs with (parameterless) procedures and synchronization [23]. Here, we sharpen this result by provin ...

Keywords: complexity, interprocedural analysis, parallel programs, slicing, undecidability

19 Evaluating explicitly context-sensitive program slicing
Gagan Agrawal, Liang Guo
June 2001 **Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering**

Full text available: pdf(197.56 KB) Additional Information: full citation, abstract, references, citings, index terms

One of the important issues in constructing interprocedural program slices is maintaining context-sensitivity or preserving calling context when a procedure is called at multiple call sites. Though a number of context-sensitive techniques have been presented in the last decade, the following important questions remain unanswered: 1) What is the level of precision lost if context-sensitivity is not maintained ? 2) What are the additional costs for achieving context-sensitivi ...

20 Automated method-extraction refactoring by using block-based slicing
Katsuhisa Maruyama
May 2001 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 2001 symposium on Software reusability: putting software reuse in context**, Volume 26 Issue 3

Full text available: pdf(174.09 KB) Additional Information: full citation, abstract, references, index terms

Refactoring improves the design of existing code but is not complex to do by hand. This paper proposes a mechanism that automatically refactors methods of object-oriented programs by using program slicing. To restructure a method without changing its observable behavior, the mechanism uses block-based slicing that does not extract the fragments of code from the whole program but from the region consisting of some consecutive basic-blocks of the program. A refactoring tool implementing the m ...

Results 1 - 20 of 105 Result page: 1 2 3 4 5 6 next